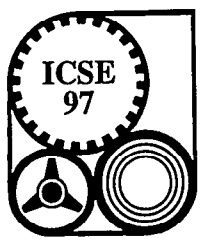
Testing and Analysis



Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1997		**00-00-1997 to 00-00-1997**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Using Formal Methods to Reason about Architectural Standards	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Virginia,Department of Computer Science,151 Engineer's Way,Charlottesville,VA,22904-4740**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**12**	

Using Formal Methods to Reason about Architectural Standards

Kevin J. Sullivan
Computer Science Department
University of Virginia
Thornton Hall
Charlottesville, VA 22903 USA
+1 804 982-2206
sullivan@virginia.edu

John Socha
Socha Computing, Inc.
13 Central Way, Suite 1200
Kirkland, WA 98033 USA
+1 206 822 9300
jsocha@socha.com

Mark Marchukov
Computer Science Department
University of Virginia
Thornton Hall
Charlottesville, VA 22903 USA
+1 804 982 2292
march@cs.virginia.edu

ABSTRACT
We present a study in which we used formal methods to reason precisely about aspects of a widely used *software architectural standard*, namely Microsoft's Component Object Model (COM). We developed a formal theory of COM to help us reason about a proposed compositional architectural style based on COM, intended for use in a novel commercial multimedia authoring system. The style combined COM objects, integration mediators, and the COM reuse mechanism of *aggregation*. Our use of formal methods averted an architectural disaster by revealing essential but subtle and counterintuitive properties of COM. We partially validated our theory by subjecting it to review by the designers of COM and by testing it against other available data. The theory has good evidential support.

Keywords
Software engineering, formal methods, partial specification, architecture, integration, mediator, Component Object Model, COM, OLE, ActiveX, empirical, Microsoft, multimedia

INTRODUCTION

The architectural designs of a vast number of systems will depend on widely used architectural standards. Today, such standards include Microsoft's Component Object Model (COM) [7] and the Object Management Group's Common Object Request Broker (CORBA) [8].

Such standards should be treated as critical infrastructure systems. Architectural standards that provide foundations for the interoperation of independent applications are especially critical, because errors resulting from improper, unanticipated or innovative use of such standards might go unnoticed until interactions among fully deployed applications finally reveal "killer" design faults.

Permission to make digital/hard copies of all or part of this material for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee
ICSE 97 Boston MA USA
Copyright 1997 ACM 0-89791-914-9/97/05 ..$3.50

Any lack of clear guidance in the proper application of an architectural standard puts adopters at an undisclosed risk of making errors in the critical, early architectural stages of system design. To the extent practicable, users should be relieved the burden of having to reason about subtle but critical aspects of such standards. The designers of such standards should specify their subtle aspects carefully, and determine their architecturally important properties. The products of these efforts should then be made available in the form of tools, documentation, or even theorems, to help users verify the legality of proposed uses of such standards.

This paper presents a study in which we used formal methods [9,19,20] to develop a theory of a *de facto* software architectural standard, namely Microsoft's Component Object Model (COM). We did this to reason effectively about the conformance of a proposed COM-based architectural style to the standard. In the absence of prior work articulating architecturally critical properties, and in the presence of what we saw as subtleties, we had to bear the burden of reasoning precisely about the standard.

At stake was Socha Computing's multimedia authoring system, Herman. Our approach averted a costly commitment to a flawed architectural style based on a combination of mediators [21,22,23], COM objects, and the COM reuse mechanism of *aggregation*.

The work described in this paper began when our initial, informal attempt to convince ourselves of the legality of the proposed style failed. Subtleties in the design of COM and the silence of the published specification on key issues made it hard to reason informally with confidence. To facilitate reasoning, we decided to capture relevant aspects of COM in the form of a mathematical theory from which we could deduce key properties of the standard.

We began by sketching a modestly rigorous theory using basic set theory concepts, based on a careful reading of the COM specification [7]. We were astonished to find that the theory predicted that our use of aggregation was illegal. We concluded tentatively that our proposed style was illegal, because, in particular, it appeared that the COM specification precluded the use of COM *aggregation* as a

compositional information hiding mechanism.

We then tested our theory by submitting our conclusion for review by the designers of COM, and by checking it against other documents [11,12,18]. We found our conclusion to be inconsistent with the designers' intentions. Our initial theory was thus not entirely correct.

We revised the theory somewhat to accommodate the new data. The revised theory led us to conclude that our proposed style was not inconsistent with the rules of COM, but that it wouldn't work as desired. We also concluded that COM is even subtler than we at first believed.

Having refined our theory, we decided to try to build additional confidence in it by increasing our level of rigor. We expressed the theory in the Z language [20], checked its syntax using the Z/Eves system [14], and proved our theorems more rigorously. While our main architectural insights emerged from the work done at a modest level of rigor, we obtained deeper insight into the precise nature of COM when we made the theory precise.

The rest of the paper is organized as follows. First, we summarize the relevant aspects of COM. Next, we present our proposed architectural style. Following that, we give a brief overview of the problems we encountered. The next section presents our formal theory, and the one after that presents our two key theorems. Next we use the theorems to reason about two COM-based architectural styles, including our own. We then summarize our results, discuss related work, and finally conclude.

COM AS AN ARCHITECTURAL STANDARD

COM is an important architectural foundation for much component-based software. As an architectural standard, COM defines the form of the components from which such systems are built, several reuse and composition mechanisms, and a set of properties that objects and compositions of objects should have. As a widely used standard it exerts an important influence on the world of real software. COM is the architectural foundation for OLE [5] and ActiveX [25], which are themselves foundations for important systems used by many individuals and by large segments of industry, government, and the military.

For our purposes, COM has several key features. First, a COM object exposes multiple interfaces. Each interface defines a set of operations for one service that the object supports: e.g., persistence, cut-and-paste, and domain-specific computation. Each interface belongs to one or more interface types, each of which is identified by unique interface identifiers *(IID)*. COM objects interact with each other solely through pointers to their respective interface instances. See Fig. 1.

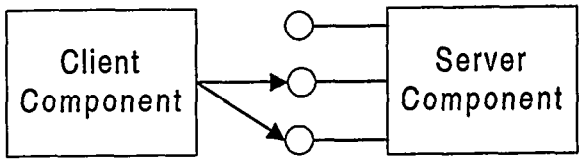


Fig. 1 Components (rectangles) expose interfaces (circles) that are accessed through pointers (arrows).

Second, every interface exports a special operation called *QueryInterface*. *QueryInterface* allows a client with a pointer to any interface on an object to obtain pointers to other interfaces on the same object. *QueryInterface* allows objects that were designed independently to negotiate communication protocols dynamically. As the basis for the interoperation of COM-based systems, *QueryInterface* is the heart of COM. "There is nothing as important to COM as *QueryInterface* [18, p. 56]." It is here, with *QueryInterface*, that we had our architectural difficulties.

In more detail, *QueryInterface* takes an *IID* as a parameter and returns, through another parameter, a pointer to an interface of the designated type on the same object. If the object does not support the designated type of interface, *QueryInterface* returns a null pointer. The return value indicates whether an interface was returned successfully.

Most of COM's reuse and composition mechanisms are traditional object-oriented design constructs. They include explicit procedure invocation, implicit invocation [14], and delegation of calls to contained objects. However, COM also provides an innovative mechanism called *aggregation*. In aggregation, one object, the *outer*, contains other objects, the *inners*. When the outer object is queried for an interface, it can return a pointer to an interface that actually belongs to an inner object.

Aggregation is useful when an inner object provides an interface whose implementation matches the one required by clients of the outer object. Aggregation permits calls made by clients to be handled by the inner without the overhead that would be required for the outer to delegate calls to the inner. In Fig. 2, two of the interfaces of the outer object are actually obtained from inner objects.

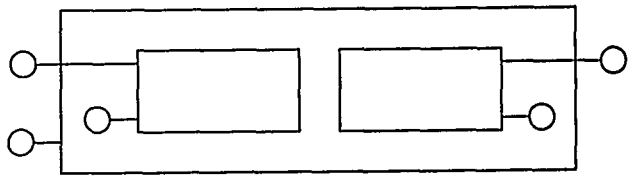


Fig. 2 An outer object aggregates two inner objects and exposes two of their interfaces to its clients

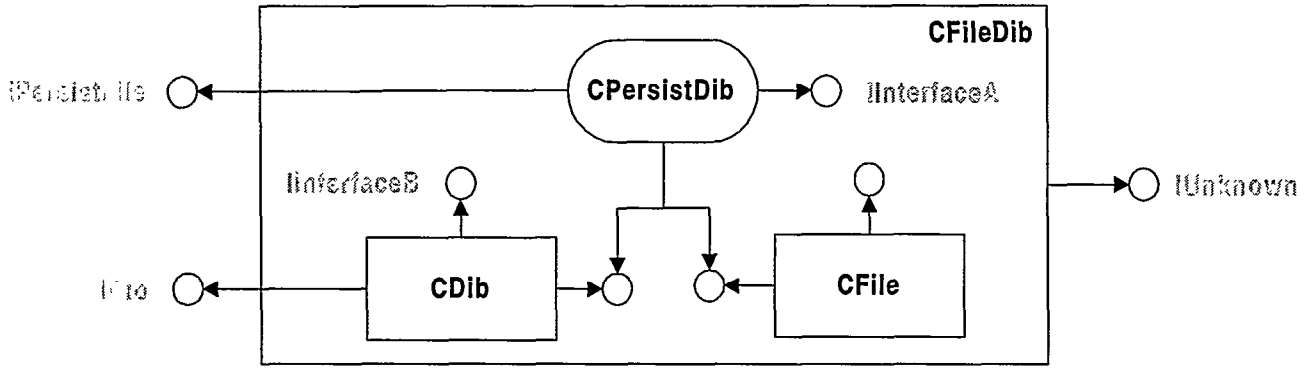


Fig. 3 Composition of two multimedia components (CDib and CFile) into a larger multimedia component (CFileDib) through the use of a mediator (CPersistDib) and COM aggregation (by CFileDib). Some interfaces of the subsystem components (IInterfaceA and IInterfaceB) are hidden by the outer object; others are exposed (IDib and IPersistFile). In addition, the outer object exports its own interface (IUnknown).

THE PROPOSED ARCHITECTURAL STYLE

The proposed architectural style for Herman was driven by two basic requirements. First, it had to support drag-and-drop manipulation of multimedia components. Second, it had to support the recursive composition of independent multimedia components into larger components.

Drag-and-drop was to be based in part on a decision to implement multimedia components as COM objects. The compositionality requirement was to be met through a combination of mediators and aggregation. Mediators [21,22,23] provided an attractive mechanism for integrating components into subsystems. Aggregation was to support abstraction of subsystems through the encapsulation and export of selected interfaces. This architectural style appeared to use COM in a simple, straightforward way.

Fig. 3 illustrates the style. The mediator (rounded rectangle) integrates the components (rectangles) into a subsystem, which is then aggregated to make a larger component (outer rectangle). The essence of the proposed approach was the selective hiding of the interfaces of the aggregated objects. By supporting the selection of variant components and mediators, such aggregates would define reference architectures [3,4] for families of related multimedia components—an interesting idea that we can't pursue further in this paper.

OVERVIEW OF THE PROBLEM

An initially minor concern for the legality of the proposed architectural style led us to try to convince ourselves of its legality. The more we worked on this, the more we realized that it was hard to reason about some non-obvious aspects of COM. Our need to understand these subtleties led us to use formal methods to build an abstract model, or theory, of the aspects of concern.

The first version of our theory indicated that our style violated the COM standard, and, that many other seemingly natural designs would, too. In particular, we disproved the putative theorem that COM aggregation supports abstraction through the selective hiding of the interfaces of aggregated objects. COM appeared not to support such abstraction. Specifically, we proved that an outer object would have to export interfaces for all types of interfaces exported by inner objects. See Fig. 3 again. Our theory said that the absence of interfaces of types *IInterfaceA* and *IInterfaceB* on the outer *CFileDib* component would be illegal. As a corollary, we concluded that our architectural style was illegal, because it depended on selective hiding of interfaces.

After a subsequent exchange with the developers of COM, we amended our theory and revised our conclusion. On the basis of our revised theory we proved two key results that we now believe to be valid. First, aggregation compromises object identity as defined by COM. In particular, the mediator in Fig. 3 would find CDib and CFile to have the same object identity. Second, although selective hiding is legal, its use implies that inner objects do not satisfy the rules for *QueryInterface*, and therefore that they cannot be treated as legal COM objects by other objects such as our mediators. Our use of COM aggregation as a composition mechanism therefore presented much more serious difficulties than we had anticipated.

FORMAL MODEL AND REASONING

We now present our revised and formalized model of the relevant aspects of COM, using basic concepts from first-order set theory, expressed in the Z language [20]. In the next section, we use this model to deduce expected properties of COM, which we model as theorems in our theory.

Interfaces

Each COM interface instance (or *interface*) belongs to at least one component and satisfies one or more interface specifications (*specifications* or *types*). Like abstract classes, specifications declare the operations of interfaces. Like concrete classes, components bind implementations to the operations declared by their interfaces. We need not discuss implementations any further. A unique interface identifier (or *IID*) identifies each specification. We model interfaces, specifications and *IIDs* as given sets of unelaborated entities in Z, as their details are irrelevant.

$$[IID, Interface, InterfaceSpec]$$

The heart and distinguishing feature of COM is a special interface type, *IUnknown*, whose *IID* is *IID_IUnknown*. *IUnknown* exports three operations. Two that we do not discuss further support reference counting for garbage collection. The third is *QueryInterface*. All COM interfaces can be viewed as inheriting from, i.e., as being polymorphic with, *IUnknown*. Thus, all COM interfaces export the *QueryInterface* operation. We formalize *IUnknown* as a specification, and *IID_IUnknown* as an *IID* in the following Z axiom.

$$\begin{array}{|l} IUnknown : InterfaceSpec \\ IID\_IUnknown : IID \\ \hline \end{array}$$

We model the association of each interface specification with its unique *IID* as a total one-to-one function that, in particular, associates *IUnknown* with *IID_IUnknown*.

$$\begin{array}{|l} IIDOfInterfaceSpec : InterfaceSpec \rightarrowtail IID \\ \hline IIDOfInterfaceSpec(IUnknown) = IID\_IUnknown \end{array}$$

We model the polymorphism of all interfaces with *IUnknown* as a relation *InterfaceSpecOf* that maps interfaces to the specifications they satisfy. The predicate, which treats the relation as a set of tuples, requires every interface to satisfy at least the *IUnknown* specification.

$$\begin{array}{|l} InterfaceSpecOf : Interface \leftrightarrow InterfaceSpec \\ \hline Interface \times \{IUnknown\} \subseteq InterfaceSpecOf \end{array}$$

Next, we use simple relational composition to define a new relation, *IIDOfInterface*, which maps each interface to the *IIDs* of the specifications that the interface satisfies. It is easy to see is that *IIDOfInterface* maps each interface to at least *IID_IUnknown*.

$$\begin{array}{|l} IIDOfInterface : Interface \leftrightarrow IID \\ \hline IIDOfInterface = InterfaceSpecOf \fatsemi IIDOfInterfaceSpec \end{array}$$

Interface Traversal

Because each interface is polymorphic with *IUnknown*, a pointer to any interface can be treated as a pointer to *IUnknown*; so *QueryInterface* can be called through any such interface. The purpose of *QueryInterface* is to allow a client with a pointer to one interface to navigate to other interfaces. In the rest of this paper we ignore the distinction between interfaces and pointers to interfaces. We thus model the *QueryInterface* operation of each interface as a partial function *QI* that maps the interface and a given *IID* to another interface.

$$\begin{array}{|l} QI : Interface \times IID \nrightarrow Interface \\ \hline \end{array}$$

The COM standard requires that it be possible to obtain an interface of type *IID_IUnknown* by calling *QueryInterface* on any interface. We represent this requirement in our theory with a predicate stating that *QI* be defined for every interface with *IID_IUnknown* as the given *IID*.

$$Interface \times \{IID\_IUnknown\} \subseteq dom\ QI$$

Components

For our purposes, a COM component is an object that exposes a finite set of interfaces. The set of interfaces exposed by an object is defined recursively. Every object exposes a distinguished interface that satisfies at least the *IUnknown* specification. In COM, this interface is called the distinguished *IUnknown* of the object. If defined, the result of applying *QI* to an interface of an object is another interface on the same object. We define the set of *IIDs* of an object to be equal to the set of *IIDs* of the specifications that are satisfied by the individual interfaces of the object.

$$\begin{array}{|l} \textit{Component} \\ \hline Interfaces : \mathbb{F}\ Interface \\ iids : \mathbb{F}\ IID \\ iunknown : Interface \\ \hline iunknown \in Interfaces \\ \forall\, i : Interface; d : IID \\ \quad \mid i \in Interfaces \wedge (i, d) \in dom\ QI \\ \quad \bullet\ QI(i, d) \in Interfaces \\ iids = IIDOfInterface\ (\!|\ Interfaces\ |\!) \\ \hline \end{array}$$

COM object identity is defined in terms of the distinguished *IUnknown* interfaces of components. The basis for identity is the requirement that every call to *QueryInterface* made through any interface of an object, with *IID_IUnknown* as a parameter, always returns the same, distinguished *IUnknown* interface of that object. The *identity axiom* of our model formalizes this requirement.

$$\forall X : Component; i : Interface \mid i \in X.Interfaces \bullet QI(i, IID\_IUnknown) = X.iunknown$$

COM defines object identity as follows: Given any two interfaces, you determine whether they are interfaces on the same object by querying for *IID_IUnknown* through each, then comparing the returned interfaces (pointers). We formalize COM object identity as a binary relation $=_{com}$. It is easy to see that $=_{com}$ is an equivalence relation.

$$\_ =_{com} \_ : Component \leftrightarrow Component$$

$$\forall X, Y : Component \bullet X =_{com} Y \iff X.iunknown = Y.iunknown$$

At the heart of COM are rules governing *QueryInterface* operations that are intended to ease inter-object interface negotiation. COM requires that the *QueryInterface* operations of an object allow clients to get from any interface on that object to any other with one call to *QueryInterface* [18].

COM thus demands that *QueryInterface* operations be what it calls reflexive, symmetric and transitive. Contrary to our intuition, and to what we believe to be common understanding, COM does not require reachability of *interfaces* one from another but only the ability to get from one *type* of interface to another.

Our initial theory modeled all interfaces as following the *QueryInterface* rules. That theory led to the conclusion, contradicted by the developers of COM, that selective hiding of interfaces was illegal.

To obtain a theory consistent with both the published specification and the stated intentions of the COM designers, we changed our theory to model those interfaces that do have the reflexivity, symmetry, and transitivity properties as a subset of *Interface* called *COMInterfaces*. Thus, elements of *Interface* no longer model legal COM interfaces alone. The first requirement on a legal COM interface is that its *QueryInterface* operations return interfaces that actually have the requested *IIDs*.

$$COMInterfaces : \mathbb{P}\ Interface$$

$$\forall i : Interface; d : IID \mid i \in COMInterfaces \wedge (i, d) \in dom\ QI \bullet QI(i, d) \mapsto d \in IIDOfInterface$$

In the following paragraphs, we formalize the *QueryInterface* rules. First, COM defines reflexivity to mean that if you have a legal COM interface *i* with type *IID_Some*, then calling *QueryInterface* on *i* for *IID_Some* must succeed. It is not required that the returned interface be *i* itself, unless *i* is the distinguished *IUnknown* and *IID_Some* is *IID_IUnknown*. Recall that *IIDOfInterface* associates an interface with all of the *IIDs* that it satisfies. We formalize the COM notion of reflexivity by stating that the domain of *QI* contains the subrelation of *IIDOfInterface* restricted to the subset of legal COM interfaces.

$$COMInterfaces \lhd IIDOfInterface \subseteq dom\ QI$$

Second, COM defines symmetry to mean that if you have a legal COM interface *i* of type *IID_Some*, and if calling *QueryInterface* on *i* with *IID_Other* succeeds in returning an interface *p*, then calling *QueryInterface* on *p* with *IID_Some* must also succeed. Informally, if you can get from here to there, you can get from there to here [18]. The subtlety, again, is that "here" and "there" refer to interface types. The formal statement encodes this property, requiring in particular that it holds for all legal COM interfaces.

$$\forall a, b : Interface; iidA, iidB : IID \mid \{a,b\} \subseteq COMInterfaces \wedge (a, iidB) \in dom\ QI \bullet a \mapsto iidA \in IIDOfInterface \wedge QI(a, iidB) = b \Rightarrow (b, iidA) \in dom\ QI$$

Finally, the COM specification defines transitivity to mean, informally, that if *QueryInterface* can get you from "here to there" and "there to somewhere else," it can get you "here to somewhere else." The formal statement is similar to those in the preceding paragraphs.[1]

[1] The specification actually gives an unorthodox definition of transitivity: informally, that you can get "from elsewhere back to here." The definition is not equivalent to the ordinary definition of transitivity, and it is not strong enough to ensure that *QueryInterface* operations have the required "anywhere-in-one-step" property. We therefore interpret the COM specification as using an erroneous definition of transitivity; and we have used the common definition in place of the unorthodox one.

$$\forall a, b, c : Interface; iidA, iidB, iidC : IID \mid \{a,b,c\} \subseteq COMInterfaces \land \{(a, iidB), (b, iidC)\} \subseteq dom\ QI \bullet a \mapsto iidA \in IIDOfInterface \land QI(a, iidB) = b \land QI(b, iidC) = c \Rightarrow (a, iidC) \in dom\ QI$$

Just as we had to distinguish legal COM interfaces, we also had to distinguish legal COM objects. We model legal COM objects as a subset of *Component* whose elements have only legal COM interfaces.

$$COMObjects : \mathbb{P}\ Component$$

$$\forall C : Component \mid C \in COMObjects \bullet C.Interfaces \subseteq COMInterfaces$$

A simple property of *QueryInterface*

At certain points in formulating our theory, we found it prudent to test it against well-known properties of COM. Although not stated in the specification, the set of interface types of an object is supposed to be closed in some sense under *QueryInterface*. Goswell asserts, "The set of interface IDs [of an object] accessible via **QueryInterface** is the same for every interface.... [11]." The following statement formalizes this property. In essence, it states that from any interface on an object the same set of interface types is accessible: namely, the set of all interface types exposed by the object. A simple proof, given in the appendix, provides support for the theory insofar as it shows that the theory makes valid predictions.

Lemma: Totality of *QI*

If C is a legal COM object, and if *iidA* is a type of an interface exposed by C, then from any interface i of C it is possible to obtain an interface of type *iidA* with one call to *QueryInterface*.

$$\forall C : Component; i : Interface; iidA : IID \mid C \in COMObjects \land iidA \in C.iids \land i \in C.Interfaces \bullet (i, iidA) \in dom\ QI$$

Aggregation

In this section, we present the final part of our theory: a model of COM aggregation. We model the containment relation imposed by aggregation, and the rules governing both the interfaces of aggregated components and the implementations of their *QueryInterface* operations: specifically the COM notions of delegating and non-delegating inner interfaces.

First, as formalized in the Z axiom below, we model component hierarchy as a relation, *Aggregates*, on components (not just on legal COM objects). This formalization is abstract, but sufficient for our purposes. We model the export by outer objects of interfaces that are provided by inner objects by requiring that for any pair of objects (*outer, inner*) in the *Aggregates* relation, at least one interface of *inner* also must be an interface of *outer*.

Next, we model what COM calls the non-delegating inner *IUnknown* of an aggregated object (the unique interface h in the axiom below). We explain the need for this interface in the next paragraph. The implementation of *QueryInterface* on this interface always returns interfaces on the inner object. The outer object uses this interface to obtain inner interfaces that will be exposed to clients.

Third, we model the COM concept of delegating inner interfaces. The COM specification requires that all interfaces of inner objects other than the non-delegating *IUnknown* delegate *QueryInterface* calls to the outer. One reason for this requirement is that *QueryInterface* operations provided by interfaces exposed to clients of the outer must return only interfaces on the outer object, whether or not those interfaces are provided by the inner object. Delegation ensures that this requirement is satisfied. We model delegation as a constraint on the *QI* function. For any delegating inner interface i we require that its *QI* function (i.e., the function obtained by fixing the first parameter of *QI* to be i) be equal to the *QI* function of one of the interfaces of the outer object.

$$Aggregates : Component \leftrightarrow Component$$

$$\forall I, O : Component \mid O \mapsto I \in Aggregates$$

sharing of at least one interface

$$\bullet\ I.Interfaces \cap O.interfaces \neq \emptyset$$

hidden non-delegating inner IUnknown

$$\land (\exists_1 h : Interface \mid h \in I.Interfaces \setminus O.Interfaces \land InterfaceSpecOf(\{h\}) = \{IUnknown\}$$

delegation of all but one *QI*

$$\bullet\ \exists o : Interface \mid o \in O.interfaces \bullet \forall i : Interface \mid i \in I.Interfaces \setminus \{h\} \bullet (\{i\} \times IID) \lhd QI = (\{o\} \times IID) \lhd QI)$$

TWO THEOREMS OF COM

We now present our two main theorems. As consequences of a theory that we believe models COM, these theorems make precise and explicit two critical architectural properties of COM that we had to understand in order to reason effectively about our proposed architectural style.

Theorem 1: COM Component Identity
If a component *outer* aggregates a component *inner* then *outer* and *inner* share object identity as defined for COM components.

$$\forall I, O : Component \bullet O \mapsto I \in Aggregates \Rightarrow I =_{com} O$$

This property of COM is not made explicit in the COM specification, but we find support for it in Goswell's cookbook [11]. A simple corollary is that all components within an aggregate share identity. The proof of this theorem is given in the appendix to this paper.

Theorem 2: Information Hiding
Let *outer* be a component that aggregates a component *inner*. If *inner* is a legal COM object, then the set of types of interfaces exposed by *outer* must include the set of types of interfaces exposed by *inner*.

$$\forall I, O : Component \mid O \mapsto I \in Aggregates \bullet I \in COMObjects \Rightarrow I.iids \subseteq O.iids$$

The contrapositive of the theorem, which is also true in the theory, states that if *outer* does not expose interfaces with *IIDs* matching those of all *inner* interfaces, then *inner* is not a legal COM object. To the best of our knowledge, this property is not documented explicitly in any description of the standard. The proof is in the appendix.

ANALYSIS OF TWO ARCHITECTURAL STYLES

In this section we analyze two architectural styles using the theorems that we have proven. First, we present additional support for our theory by showing that the predicted properties of COM are not problematical when COM is used in the traditional COM style. Then we discuss the difficulties that we faced when we tried to use aggregation in our innovative architectural style.

The OLE Container and Control Style

The traditional usage of COM aggregation is called the OLE control and container idiom. In this style, an outer component wraps an aggregated component, usually an OLE control [5]. In the simple case, the outer exposes all of the interfaces of the inner except for its non-delegating *IUnknown*. The outer provides additional interfaces supporting additional services. For example, the outer might add an interface allowing the inner *control* object, such as a button, to be managed by a *container* object, such as a Visual Basic form [24]. The added interface would support placement-on-form information. In other cases, the outer hides some inner interfaces.

In both cases, the component identity theorem tells us that the outer and inner components share identity. This merging of identity creates no serious architectural problems because the only client of the inner is the outer, which treats the inner as a hidden implementation detail.

Nor does the information hiding property of COM present a problem. If the outer exposes all interfaces of the inner, then by definition it exposes interfaces with all of the types of the inner interfaces, and so the status of the inner object as a legal COM object is not necessarily compromised. If the outer hides some inner interface types, the inner is not a legal COM object; but, again, the consequences are limited because the outer treats the inner as a hidden implementation detail to which it has exclusive access.

Because the traditional use of aggregation in a control and container style does not conflict with our theory, it is not surprising that this common usage has not revealed the architectural subtlety of aggregation in general. We view the compatibility of our theory with the common usage of COM as further evidence supporting the theory.

Our Proposed Architectural Style

The key difference between the traditional usage of COM aggregation and our proposed usage is that in our style aggregated components will have clients, such as mediators, other than the aggregating outer components. See Fig. 3. Our theory revealed this proposed architectural style to be untenable.

First consider object identity. If a mediator within an aggregate compares apparently distinct but aggregated components for identity, it finds them to be identical. The loss of object identity within an aggregate can be a serious matter when what is aggregated is a subsystem having multiple, interrelated components.

Second, consider information hiding [16,17]. We find ourselves on the horns of a dilemma. Either the outer object exports interfaces whose types include all of those of all aggregated inner components, including interfaces intended solely to support the integration of the component parts; or our mediators cannot assume that the components that it mediates are legal COM objects.

In both cases information hiding is compromised. In one case, selective hiding of subsystem interfaces is precluded, and thus so is abstraction of the aggregated subsystem. If selective hiding is employed, then the mediators can no longer treat the objects that they mediate as legal COM objects because they will not follow the *QueryInterface* rules. In light of Rogerson's pithy remark, that "There is nothing as important to COM as *QueryInterface* [18, p. 56]," we have to view failure to follow the rules of *QueryInterface* as an architecturally serious matter.

The second information hiding problem bears additional discussion. It has two aspects. First, as Parnas notes in "Designing Software for Ease of Extension and Contraction," information hiding is a general concept in that "... as far as possible, even the presence or absence of a component should be hidden from other components [17, p. 229]." However, the presence of an outer is not hidden from mediators, because aggregation compromises the architectural properties of the mediated objects.

Second, because mediators can not depend on mediated components having COM-defined architectural properties, it is necessary to have ad hoc rules for aggregated objects. Given that our architectural style depends on selective hiding of inner interfaces, we decided that we had to require designers of Herman components to follow such rules. Our use of formal methods to reason precisely about the COM standard led us to change our architectural style.

SUMMARY OF RESULTS

We present a number of results. First, we developed a formal theory of subtle aspects of a widely used software architectural standard. The theory might be of considerable value to practitioners. In particular, it provides a basis for documenting subtle but important aspects of COM. Second, although our architectural style appeared to be a natural, compositional use of COM, we showed through formal reasoning that it was much more problematical than it appeared to be at first. Third, this discovery helped us to avoid a serious architectural design error in a commercial development project before it harmed the firm. Fourth, we demonstrated the profitable use of formal methods "in-the-small," not for requirements specification but to help us to reason about one difficult architectural design problem. Finally, in the methodological dimension, we have emphasized the role of an empirical approach to developing formal theories of architectural standards. When imposed on the world of software, standards impose interesting, stable structures that are amenable to empirical scientific study, with the potential for interesting results.

Having demonstrated that COM has subtle but critical architectural properties, we believe we have built a case for treating widely used architectural standards as critical infrastructure systems. The lack of a characterization of important but subtle properties shifts significant, undisclosed costs and risks onto adopters of such standards. Our case focused on the use of formal methods to make architectural properties of such standards explicit; other applications of formal methods are clearly possible, too, such as identifying ambiguities and inconsistencies.

RELATED WORK

Related work falls in several categories, especially software architecture and formal methods. At the intersection of these areas are results such as those of Abowd et al. [1], Luckham [13] and Garlan and Shaw [10]. Our work is most closely related to, and most influenced by, that of Abowd et al. and Garlan and Shaw.

Formalizing Software Architecture

Abowd et al. provide a comprehensive framework within which a broad range of abstract architectural descriptions can be given precise semantics, enabling analysis and comparison of abstract architectural styles. The authors observe that formalizing architecture can help designers to ask and answer interesting questions about such styles.

We agree. By way of contrast, our objective was not a generalized approach to formalizing descriptions in a range of styles. Rather, we were driven to a minimal use of formalism because we found that without it we could not confidently answer key questions about our specific, concrete architectural style: Was it legal with respect to the specific, widely used standard on which it was based?

Our theory makes no attempt to model dynamic semantics. We didn't have to do that to answer our particular questions. Both Luckham and Garlan model dynamic semantics at the architectural level: as event orderings in Rapide, and fair scheduling in pipe and filter architectures, for example. Nor does our theory contain any generalized concept of *connectors*. The only inter-component relation that we model is aggregation. We have tried to use the minimal sufficient formalism.

On the other hand, our theory appears to be richer than that of Abowd et al. in other dimensions. We model a complex relation over interfaces (corresponding to ports in Abowd et al.), namely the *QueryInterface* relation. We also model the effect on this relation of a separate relation between components, namely aggregation. In Abowd et al., there appears to be no corresponding concept of either relationships among ports, or of interactions between inter-port relationships and inter-component relationships, such as aggregation.

Despite the differences in focus and generality, our work overlaps with that of Abowd et al. on one important issue: the compositionality of architectural styles. Abowd et al. show that within their theory, pipes in pipe and filter architectural styles are compositional, but that components that announce events in implicit invocation styles are not compositional. We showed that legal COM objects are not compositional under aggregation.

It is not surprising that we both focus on compositionality. It is a critical property of any architectural style, because it greatly facilitates the construction of new systems from existing parts. Complications in compositionality are thus of deep concern to the architect. In this paper, we

characterize previously undocumented complications in the compositionality properties of an important and widely used architectural standard.

Formal Methods and Software Standards

Our concern for the integrity of standards is not new. Nor is it surprising that subtleties that have not been analyzed can have astonishing consequences. Ardis et al. noted that ambiguities in the specification of a telecommunications protocol make it "...possible to completely defeat the protection switching protocol, causing the communication link to fail, even though there was at least one working line in each direction [2]." They then express the hope that "...future authors of standards will consider using formal languages, so that ambiguity can be minimized."

Our experience lends support to Ardis et al. Formal methods appear to have an important role to play in validating widely used standards. Until a standard has been subject to rigorous analysis, however, and the results of the analysis made available to users, the costs of reasoning about the conformance of designs to the standard and the risks of not doing so are shifted onto its users. Our experience shows that it is possible for even a small firm to use formal methods profitably, "in-the-small," with modest coverage and rigor, to reason about difficult design issues.

Object Models

Bryant and Evans [6] discuss a formalization of the CORBA object model in Z; but neither the model or any of its consequences is presented in detail. The purpose of the specification that they emphasized was to help resolve ambiguities and inconsistencies in the specification to facilitate negotiation among those defining the standard. The extent to which this work has progressed is unclear.

CONCLUSION

We have developed a formal theory of certain aspects of the COM architectural standard. From the theory we deduced subtle and counterintuitive but architecturally important properties of COM. An initial, modestly formal theory was sufficient to reveal problems in the proposed architectural style for the Herman multimedia-authoring environment. A subsequent test of the theory indicated that our conclusions were not precisely correct; however, the required changes did not fundamentally change our conclusions about COM or our proposed architectural style.

To build additional confidence in the amended theory, we decided to express it more precisely. We formalized the initial, back-of-the-envelop theory by writing it in Z, checking its syntax using the Z/Eves theorem prover, and expressing and proving our main theorems rigorously. We have not yet verified our proofs mechanically.

Socha Computing benefited most from our early, modestly rigorous analysis, and from the feedback obtained when we subjected our initial conclusions to review. Nevertheless, our subsequent emphasis on increasing rigor was useful. Like COM, we found our theory itself to be unexpectedly subtle. The more aggressive use of formalism led us to a theory that is more convincing to us and in which we thus have significantly greater confidence.

While our discoveries about COM came as surprises, Socha Computing, Inc. has not changed its decision to use COM for its Herman system. In fact, COM is being used as aggressively as at first envisioned. We find the design of COM elegant and innovative. COM is the object standard for many applications in very wide use. COM isn't broken.

Nevertheless, COM and comparable standards do define the foundations of vast numbers of important programs. These standards are thus important infrastructure systems, and should be specified with commensurate care. The presence and the implications of any major subtleties must be explicated. We used formal methods at varying levels of rigor, profitably, to reason about one such architectural standard and to verify the conformance of a design to it.

As to who should bear the costs of reasoning about such subtleties, and the risks of not reasoning, that is an extra-scientific issue to be resolved by extra-scientific means: the market, policy, etc. We would prefer to see the designers of such standards perform the necessary reasoning, so as not to expose adopters to undisclosed risks. Perhaps the research community can influence the incentive field by subjecting real systems and real standards to scientific study.

ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under grants CCR-9502029 and CCR-9506779, and by the Defense Advanced Research Projects Agency (DARPA) under grant F30603-96-1-0314. We gratefully acknowledge the contribution of Tony Williams, of Microsoft Corporation, who agreed to criticize our early theory of COM. We thank Odyssey Research Associates for the use of the Z/Eves system. We thank the anonymous reviewers, who encouraged more rigorous formalization of our theory, and who contributed the term "in-the-small" as we have used it in this paper. Finally, we thank David Garlan for support and guidance.

APPENDIX: PROOFS OF THEOREMS

Proof of Totality Lemma:

Suppose that

$C : Component \mid C \in COMObjects$
$i : Interface \mid i \in C.Interfaces$
$iidA : IID \mid iidA \in C.iids$
Then
$\exists\, a : Interface \mid a \in C.Interfaces$
$\bullet\ a \mapsto iidA \in IIDOfInterface$
By the identity axiom of QI
$$QI(i, IID\_IUnknown) = QI(a, IID\_IUnknown) = C.iunknown$$
By symmetry of QI $(C.iunknown, iidA) \in dom\ QI$
Then by transitivity of QI $(i, iidA) \in dom\ QI$
□

Proof of COM Identity Theorem:
Suppose that
$I, O : Component \mid O \mapsto I \in Aggregates$
Then by an axiom of aggregation
$\exists\, z : Interface \bullet z \in O.Interfaces \cap I.Interfaces$
Thus by the identity axiom we have
$I.iunknown = QI(z, IID\_IUnknown) = O.iunknown$
And so $I =_{com} O$.
□

Proof of Information Hiding Theorem:
Let $I, O : Component$
$\quad \mid I \in COMObjects \wedge O \mapsto I \in Aggregates$
Let $iidX : IID \mid iidX \in I.iids$.
We shall show that $iidX \in O.iids$.
By Identity Theorem
$$I.iunknown = O.iunknown$$
Since $iidX \in I.iids$,
$$\exists\, x : Interface \mid x \in I.Interfaces$$
$$\bullet\ x \mapsto iidX \in IIDOfInterface.$$
By Identity Axiom
$$QI(x, IID\_IUnknown) = I.unknown = O.iunknown$$
Note that $O.iunknown$ is in $COMInterfaces$ since it is also an interface of a legal COM object I.
By symmetry, $(O.iunknown, iidX) \in dom\ QI$.
Consider $x_1 : Interface \mid x_1 = QI(O.iunknown, iidX)$.
Since $O.iunknown \in O.Interfaces$, $x_1 \in O.Interfaces$ by definition of *Component* type.
Also, $x_1 \mapsto iidX \in IIDOfInterface$ because
O.iunknown $\in COMInterfaces$ and by the definition of $COMInterfaces$
$$QI(O.iunknown, iidX) \mapsto iidX \in IIDOfInterface.$$
Thus $iidX \in IIDOfInterface(\!|\, O.Interfaces \,|\!) = O.iids$.
□

REFERENCES

1. Abowd, G., R. Allen and D. Garlan "Formalizing style to understand descriptions of software architecture," *ACM Transactions on Software Engineering and Methodology*, vol 4, no. 4, Oct 1995, pp. 319-364.
2. Ardis, M.A., J.A. Chaves, L.J. Jagadeesan, P. Mataga, C. Puchol, M.G. Staskauskas and J. Von Onnhausen, "A framework for evaluating specification methods for reactive systems," *Transactions on Software Engineering*, vol. 22, no. 6, June, 1996, pp. 378-389.
3. Batory, D. and S. O'Malley, "The design and implementation of hierarchical software systems with reusable components," *ACM Transactions on Software Engineering and Methodology* vol. 1, no. 4, pp. 355-398, Oct. 1992.
4. Batory, D., L. Coglianese, M. Goodwin and S. Shafer, "Creating reference architectures: an example from avionics," Proceedings of SSR'95, *Software Engineering Notes*, April 28-30, 1995, pp. 27-37.
5. Brockschmidt, K., *Inside OLE*, Microsoft Press, 1996.
6. Bryant, T. and A. Evans., "Formalizing the object management group's core object model," *Computer Standards and Interfaces*, vol. 17, no. 5-6, pp. 481-9, September 30, 1995.
7. Common [sic] Object Model Specification, Microsoft Developer Network Library, Microsoft Corporation, 1996 (especially sections 3.3.1 and 6.6.2).
8. *The Common Object Request Broker: Architecture and Specification*, Object Management Group, Inc., 1995
9. Craigen, D., S. Gerhart and T. Ralston, "An international survey of industrial applications of formal methods, Volumes 1 and 2," NIST GCR 92/626, U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, MD, March, 1993.
10. Garlan, D. and M. Shaw, "An introduction to software architecture," *Advances in Software Engineering and Knowledge Engineering*, Vol. 1, World Scientific Publishing, 1993.
11. Goswell, C., "The COM Programmer's Cookbook," Microsoft Office Product Unit, Spring 1995, revised September 13, 1995, Available on the World-Wide Web at the time of submission of this paper through http://www.microsoft.com/oledev/olecom/com_co.htm
12. Kindel, C., "The rules of the component object model," Microsoft Developer Network Library, Technical Articles: Windows: OLE COM, Microsoft Corporation, October 20, 1995.

13. Luckham, D.C., J.J. Kenney, L.M. Augustin, J. Vera, D. Bryan, and W. Mann, Specification and Analysis of System Architecture Using Rapide, *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 336-355, Apr 1995.

14. Meisels, I. And M. Saaltink, *The Z/EVES Reference Manual (for Version 1.3), Draft,* ORA Canada Technical Report TR-96-5493-03b, December 1995, revised November 1996.

15. Notkin, D., D. Garlan, W.G. Griswold, and K. Sullivan, "Adding Implicit Invocation to Languages: Three Approaches," *Object Technologies for Advanced Software*, Lecture Notes in Computer Science, Vol. 742, pp. 489-510, Nov 1993.

16. Parnas, D., "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, December, 1972, pp. 1053-1058.

17. Parnas, D. "Designing software for ease of extension and contraction, " Proceedings of the 3rd International Conference on Software Engineering, pp. 264-277, IEEE Computer Society Press, 1978

18. Rogerson, D., *Inside COM*, Microsoft Press, 1997

19. Rushby, J., *Formal Methods and Digital Systems Validation for Airborne Systems,* NASA Contractor Report 4551, National Aeronautics and Space Administration, Office of Management, Scientific and Technical Information Program, 1993.

20. Spivey, M., *The Z Notation: A Reference Manual,* Prentice-Hall, 1992

21. Sullivan, K.J., "Mediators: Easing the Design and Evolution of Integrated Systems," Ph.D. Dissertation, University of Washington Department of Computer Science, August 1994.

22. Sullivan, K.J. and D. Notkin, "Reconciling Environment Integration and Evolution," ACM *Transactions on Software Engineering and Methodology* vol. 1, no. 3, July 1992.

23. Sullivan, K.J., I.J. Kalet and D. Notkin, "Mediators in a Radiation Treatment Planning System," IEEE *Transactions on Software Engineering*, vol. 22, no. 8, August 1996, pp. 563-579.

24. *Visual Basic 4 User's Manual*, Microsoft Corporation, 1996.

25. Williams, A., *Developing ActiveX Web Controls*, Coriolis Group, Inc, 1996